Communication & Cognition
Vol. 58, No. 1 - 2 (2025), pp. 3 - 22

IMPLEMENTING AND ASSESSING RETRIEVAL AUGMENTED GENERATION (RAG) FOR LLM-BASED DOCUMENTS QUERIES

Peter Kaczmarski and Fernand Vandamme

https://doi.org/10.57028/C58-003-Z1079

Abstract

In recent years, AI-related technology referred to as RAG (Retrieval Augmented Generation) (Lewis, 2020) gained a lot of attention. In the RAG-approach, custom sources of information are used to seed the knowledge obtained from a LLM (Large Language Model), thus forming an approach which solves the issue of adapting the LLM to cope with custom external information. Using the RAG-scenario, various information processing use cases can be implemented, such as AI-based document management, AI-enhanced web search, AI-based online service support, etc. This paper outlines the main components of the RAG-workflow such as chunking and embedding the input documents, as well as similarity search, and LLM -based user query processing. The RAG approach is illustrated via Python implementation which is used to validate the procedure via a simple example of processing a multi-topic document. Experimental results are discussed showing the feasibility of this approach, as well as illustrating the need for further research and enhancements, for example by the use of the RAPTOR concept (Sarthi, 2024).

Keywords

AI, Artificial Intelligence, chunking, Chroma, document management, embedding, image analysis, James Joyce, Large Language Model, LLM, OpenAI, Python, q&a, RAG, Retrieval Augmented Generation, RAG enhancements, RAPTOR, scenario analysis, speech to text, Theory of Mind (ToM), Ulysses, vector database.

1. Introduction

In this paper, we focus on integration of Large Language Models (LLMs) with procedures for AI-based document search and retrieval. The applications of LLMs received recently a lot of attention (see for example (Kaczmarski & Vandamme, 2024), and, in this context, the topic of AI-based search has a significant potential to improve document management approaches in general. The major challenge in applying Large Language Models (LLMs) in searching the content of custom/private document collections is that the language models such gpt-4o, phi-3, or llama-3 have no knowledge of these documents as they are trained on public documents only. However, a standard approach, commonly known as RAG (Retrieval Augmented Generation) has recently been developed to alleviate this issue (OpenAI, 2024) (Grainger, 2024) (Wikipedia, 2024a) (Lewis, 2020).

The focus of this paper is on introducing the common RAG procedure, presenting its implementation and features, and also on testing its feasibility. Such a common RAG approach is based on a few steps including document chunking, embedding, indexing, and searching by the use an LLM. In this approach, custom documents to be searched are first indexed in a special way compatible with LLM technologies. This process generates embeddings (Wikipedia, 2024b), and these embeddings are used to match user query with most relevant documents. Next part in the RAG-based search is to provide the retrieved content to a Large Language Model so that the LLM reasoning can be applied to this content.

The above RAG procedure is highlighted in Section 2, and Section 3 illustrates Python implementation of the main steps involved.

Section 4 describes experimental results of testing the RAG approach by the use of a multi-topic document which allows to study the feasibility of this approach of content search. There are 3 distinct topics included in the test document:

- discussion about ‘Ulyssess’ of James Joyce, which is of one of the most difficult books in English literature

- discussion of Theory of Mind tests (Kaczmarski & Vandamme, 2024) (Kosinski, 2023a)
- discussion of image analysis by LLMs (Kaczmarski & Vandamme, 2024)

The aim of this test is to study the usefulness of RAG in identifying distinct topics in input documents and in discussing them with the LLM.

In Section 5, test results are analyzed, and the main attention points are related to the strengths and weaknesses of the tested version of the RAG procedure.

The final section (Section 6: Conclusions) summarises the status of the RAG technology and discusses the directions of research and enhancements of the RAG procedure.

2. RAG Procedure Summary

The major steps of the RAG procedure are shown in Fig.1. The procedure has 2 main flows

- document processing flow
- user queries flow

As shown in Fig. 1, document processing flow consists of input documents chunking, generation of embeddings and saving the embeddings in a vector database. User queries flow starts with creation of a query, and generation of embeddings. Next, these embeddings are used in similarity search in the database via Retrieval module, and the retrieved chunks are merged with user query and sent to the LLM. There is a loop which allows the user to enter next query or to stop the search.

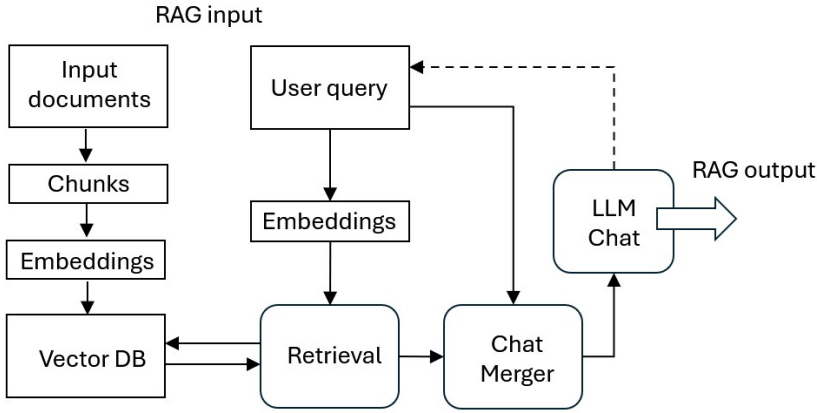

Fig 1. Major steps within the standard RAG procedure.

3. Implementation Overview

The Python implementation (full code is presented in Appendix A) follows the standard RAG procedure flow model shown in Fig.1. The major steps of the procedure are described below.

On the side of document indexing, document reading is the first step, and it is implemented by the following code:

```
loader = TextLoader('rag-test-2.txt')
documents = loader.load()
```

Document loaders for this part are provided by the library:

```
langchain_community.document_loaders
```

Next stage is document chunking which uses a library function RecursiveCharacterTextSplitter (from the library langchain.text_splitter):

```
text_splitter = RecursiveCharacterTextSplitter(
chunk_size=800,
chunk_overlap=100)
chunks = text_splitter.split_documents(documents)
```

In the next stage, embeddings are computed for each chunk using OpenAIEmbeddings. First, an object is configured which is used to generate an embedding for any chunk.

```
embeddings_creator_open_ai = OpenAIEmbeddings()
embeddings_creator_open_ai.model =
'text-embedding-3-small'
embeddings_creator_open_ai.dimensions = 1536
```

Next, the corresponding embeddings are created for each chunk and saved in a local vector store, in this case in a Chroma database.

```
vector_store_chroma=Chroma.from_documents(
chunks,
embeddings_creator_open_ai,
persist_directory="./chroma_db")
```

Concerning search and retrieval, entering a query by the user forms the first step, and it is implemented in the following way.

```
while True:
user_query = input('User query (or Quit to exit) =')
if user_query == 'Quit':
break
#
```

Next step is to generate the embedding for the entered query and to perform a similarity search in the vector database.

```
user_query_embedding = embeddings_creator.embed_que
ry(user_query)
raw_sim_result = vector_store.similarity_search_by_vector(
user_query_embedding,
3)
```

A simple merging approach is used to combine user query with retrieved chunks and send the result to the LLM. The LLM specified here is a version of OpenAI gpt-4 (gpt-4-turbo-2024-04-09).

```
content_all=''
for x in raw_sim_result:
content_all += x.page_content
content_all += '\r\n---\r\n'
```

```
user_query += "++++ " + content_all
chat_list.append({"role": "user", "content": user_query})
response = openai.chat.completions.create(
model="gpt-4-turbo-2024-04-09",
messages=chat_list, # sends each time full chat_list
temperature=1.0
)
```

Next stage is reading the LLM reply by the user and sending a new query. At this point, the user is at the start of the queries loop, and he can decide to quit the session or to enter next question.

```
while True:
user_query = input('User query (or Quit to exit) =')
if user_query == 'Quit':
break
#
```

For more details, refer to full Python code in Appendix A.

4. Experimental Results

The implementation presented in Section 3 has been used to process a text document reproduced in Appendix B, which has been written on purpose as a short and easy to follow multi topic document thus allowing to query the RAG system about the topics contained therein. The main part of this Q&A experiment has been not only to check LLMs capability to identify the presence in the text of topics suggested by the user but also to provide related knowledge beyond the factual knowledge present in the document. As already mentioned, the LLM used in this test is a version of OpenAI gpt4 (gpt-4-turbo-2024-04-09). The complete Q&A session is presented below.

User query nr 1:

Please summarize the topic of image analysis described in the document.

AI: The third use case in the document focuses on implementing LLM-based image analysis, specifically using the OpenAIs LLM GPT-4o model. This model was publicly released in May 2024. The implementation involved analyzing

an auditorium image using a modified Python script based on Appendix A and guided by the OpenAI Vision API's online documentation. The document presents both the prompt used and the model's response to demonstrate this use case.

User query nr 2:

Using the attached information, please summarize the topic related to the book Ulysses.

AI: The first use case discussed in the document involves the book "Ulysses" by James Joyce. It is focused on evaluating and comparing the performance of different LLMs using a set of three questions:

1. Is "Ulysses" by James Joyce difficult to read?

2. When was it written?

3. Is it possible to summarize the book in no more than 150 words?

These questions are used to assess the capabilities of LLMs in understanding and processing information about complex and historically significant literary work.

User query nr 3:

Based on the attached information, please summarize the presented information about Theory of Mind.

AI: The document discusses the Theory of Mind (ToM), a concept developed over the past 30 years in child psychology, highlighted by Cherry and Ruhl in 2023. To illustrate a ToM test, the document presents a scenario from a study by Kosinski in 2023. The scenario involves John, Mark, a cat, a box, and a basket. John puts the cat in the basket and then leaves; Mark moves the cat from the basket to the box and leaves. The test question asks where John will look for the cat upon returning, to assess if participants understand that John lacks the knowledge of the cat being moved to the box. This scenario is designed to

evaluate an individual's understanding of others' beliefs and perceptions, central to Theory of Mind.

User query nr 4:

Concerning Theory of Mind scenario, please provide the answer to the question where will John look for the cat.

AI: John will look for the cat in the basket because that is where he last saw it, and he is unaware that Mark moved the cat to the box while he was away. This demonstrates John's belief or mental state about where the cat should be, which is the focus of the Theory of Mind test.

User query nr 5:

It is not clear to me who is being tested in this scenario, is it John or someone else ?

AI: In the scenario provided, John is not the one being tested; instead, the test targets the person answering the question. It evaluates whether the respondent understands John's perspective and mental state—specifically, that John does not know the cat has been moved from the basket to the box by Mark. The test assesses the respondent's Theory of Mind capability, that is, their ability to comprehend and attribute distinct mental states to others, which might differ from their own knowledge or perception.

User query nr 6:

Indeed it seems logical, many thanks ... I have to go now, cheers!

AI: You're welcome! I'm glad I could assist you. If you have more questions in the future, feel free to reach out. Have a great day and take care! Cheers!

In the above discourse, user queries number 4 and 5 are of particular interest as they go beyond the factual knowledge within the document. In this particular case, question 3 is intended to check if the

model is capable to reproduce the factual knowledge about Theory of Mind enclosed in the document and the model provides indeed an excellent reply. Nevertheless, questions 4 and 5 are still more interesting because the go beyond document content.

Concerning question 4, for instance, the document does not explain where will John look to find the cat, and this question is asked to the LLM which explains correctly that John will look into the place where he left the cat. This way the LLM provides information from beyond the considered document. Additionally, the LLM seems to recognize human mental state in this ToM scenario.

Question 5 is equally interesting as it simulates a user who is confused about the object being tested ("is it John or someone else"). Therefore, this is another example of meta knowledge related to the factual content. Also in this case, the LLM gives a perfect reply, clarifying to the user that it is not John who is being tested but rather the person who replies to the question asked at the end of this test.

5. Discussion of Limitations and Challenges

Previous section has shown two capabilities important for document management which are provided by the RAG approach. The first one is that any factual information included in text chunks can be retrieved by the user by querying the RAG system. The second one is that any retrieved information can be discussed further thus revealing to the user the knowledge not contained explicitly in de retrieved document. A good example is Theory of Mind scenario description tested in section 4. In this case the information where will John look first after returning to the room is not provided in the document, yet the RAG system provides a correct response to the user.

However, there is a limitation in this particular RAG approach. The following Q&A session illustrates the problem:

User query:

What are the main topics in the attached document?

AI: The main topics in the attached document include:

1. The observation of significant differences in responses among models, particularly in response to questions 1 and 2.

2. A possible explanation for these differences, attributing them to the unconstrained nature of questions 1 and 2.

3. The mention of less variance in responses to question 3, which had a word limit constraint, although still showing some variation among the three models.

We notice that the reply from AI is in this case quite nonsensical, as it does not mention 3 use cases which form the core of the document. The main reason for this is the rigid sequence of the RAG procedure used which applies at first stage similarity search between the query and the chunks extracted from the document. Obviously, in this case there is no chunk which mentions the topics of the document. This way the RAG approach produces some spurious noise thus leading to a nonsensical reply of the LLM.

To complement the above observation, we conclude that further research is needed to investigate RAG more extensively and identify any other potential shortcomings, see for instance (Grainer, 2024). Also, RAG extensions such as RAPTOR (Sarthi, 2024) need to be studied to understand their impact on alleviating RAG drawbacks.

6. Conclusions

The RAG approach discussed in this paper forms yet another important area of applications for LLMs, which complements already well-known use cases, such as image analysis, reasoning on images, document summarisation, analysing Theory of Mind (ToM) scenarios, and more, see for example (Kaczmarski & Vandamme, 2024).

Our paper shows the major building blocks of the RAG approach and confirms their usability within the process of identifying factual, explicit knowledge present in documents, this under the condition that it is the user who initiates the search for such concrete knowledge. In other words, the user should know the content to some extent before asking questions about it. The pattern which works very well for this basic type of RAG can be summarized as “Tell me about

X" where X is assumed to be present in the imported documents. In this case, useful information can be obtained. Moreover, integration of a Large Language Model in the chat loop has a tremendous added value as any other knowledge related to "X" can be presented by the model. Therefore, the RAG use case "Tell me about X" seems to be a great added value with respect to classic indexing systems, which have the same use case being index search for a term "X" but are not able to provide a wide spectrum of knowledge associated to "X" in such a powerful way as LLMs are capable of. Therefore, we included an example of discussing the knowledge related to a topic present in the content, such as Theory of Mind, by asking questions about information which is not present in the discussed content, in this case: where will John look for the cat and who is really tested in this particular ToM scenario.

However, the expectations of the user will not be met, when asking a RAG an open question like "Tell me the topics present in documents". In this case, the standard RAG approach will give a nonsensical reply as there are no chunks in the vector database which can be associated with such question. This is implied by the limited capability of the import procedure, which is not prepared to generate this kind of related contents.

In conclusion, our discussion of RAG complements earlier publications such as (Grainer, 2024) and shows that the RAG approach, although very promising, still remains a research area which offers further improvement opportunities, see for example the RAG RAPTOR enhancement (Sarthi, 2024)

APPENDIX A:

Python RAG implementation

The Python program "main.py" presented below has been used in testing the RAG procedure.

```
# main.py
# pip install python-dotenv
```

```
# pip install langchain_community
# pip install langchain_openai
# pip install chromadb
import os
from dotenv import load_dotenv, find_dotenv
load_dotenv(find_dotenv())

# testing dotenv
currentOpenAIApiKey = os.getenv('OPENAI_API_KEY')
print('OpenAI key in use:')
print(currentOpenAIApiKey)

# libraries
from langchain_community.document_loaders import
TextLoader
from langchain.text_splitter import
RecursiveCharacterTextSplitter
from langchain_community.vectorstores import Chroma
from langchain_openai import OpenAIEmbeddings
import openai

loader = TextLoader('rag-test-2.txt')
documents = loader.load()

# split
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=800,
    chunk_overlap=100)
chunks = text_splitter.split_documents(documents)

# check chunks
print("Len:")
print(len(chunks))
for chunk in chunks:
    print("---")
    print(chunk.page_content)

# embeddings creator
embeddings_creator_open_ai = OpenAIEmbeddings()
embeddings_creator_open_ai.model = 'text-embedding-3-
small'
embeddings_creator_open_ai.dimensions = 1536

# build Chroma vector store from document chunks
vector_store_chroma=Chroma.from_documents(
```

```
    chunks,
    embeddings_creator_open_ai,
    persist_directory="./chroma_db")

print("RAG test start")

#-------------------------------------------------------------

# implementing chat function
def user_qa_chat(vector_store, embeddings_creator):
    # create chat list object
    chat_list = list()
    assistant_style = ("You are a helpful assistant." +
        "Please answer the questions in a concise way " +
        "using the attached information " +
        "behind 4 plus signs")
    chat_list.append({"role": "system", "content":
    assistant_style})
    while True:
    user_query = input('User query (or Quit to exit) =')
    if user_query == 'Quit':
    break

    user_query_embedding =
embeddings_creator.embed_query(user_query
    raw_sim_result = vector_store.similarity_search_by_vect
    or(
    user_query_embedding,
    3)

size = len(raw_sim_result)
print('raw_sim_result size:')
print (size)

content_all=''
for x in raw_sim_result:
    content_all += x.page_content
    content_all += '\r\n---\r\n'

user_query += "++++ " + content_all
chat_list.append({"role": "user", "content": user_query})
response = openai.chat.completions.create(

    model="gpt-4-turbo-2024-04-09",
    messages=chat_list, # sends each time full chat_list
    temperature=1.0
```

```
    )

    # extract the assistant's role and response
    choice = response.choices[0]
    ai_role = choice.message.role
    ai_answer = choice.message.content

    # collect AI answer in chat_list
    # as this context is needed by the model
    chat_list.append({"role": ai_role, "content": ai_answer})
    print("AI:")
    print(ai_answer)

    return

    # ------------------------------------------------------------

    # document q&a
    user_qa_chat(vector_store_chroma,
    embeddings_creator_open_ai)
    print("RAG test completed")
```

APPENDIX B:

Test document contents

This Appendix shows full content of the document 'rag-test-2.txt' used in the evaluation of the discussed RAG Python implementation. It describes three test cases for LLMs and these three descriptions form the topics discussed in the Q&A session with the RAG procedure.

File ‘rag-test-2.txt’:

First use case: discussion of a book “Ulysses” by James Joyce

A set of 3 questions has been prepared for evaluation and performance comparison of the LLMs.

- Is the Ulysses book written by James Joyce difficult to read?
- When was it written?

– Is it possible to summarize this book in no more than 150 words? Please try to do so.

The use of the novel +'Ulysses' written by James Joyce is on itself interesting in LLM context, since it is known as one of the best and most challenging literature texts ever written (Punter, 2011). However, the questions in Table 1 are on purpose somewhat fuzzy and rather basic. These questions form a good example of a stateful chat interaction (aspect number 3 from the requirements list), as question (2) refers to the book from question (1) using the reference "it" and question (3) uses 'this book'- for the same purpose.

The raw results of the conversations with each model based on the above questions are shown in Appendix B. A first observation, perhaps a surprising one, is, that the responses are quite different, in particular regarding questions 1 and 2, but also to some extent, question 3. A possible explanation of these big differences in model replies on questions 1 and 2 is that the questions were unconstrained, and this is in contrast with question 3 which constrains each model to use up to 150 words. Even for question 3 there is a lot of variation between 3 models, but the size of their replies remains limited as specified in the question.

Another observation is that all 3 models produced 'out-of-the-box' very logical and well-formulated responses with high relevancy to the content of each question. Also, the models perfectly reacted to cross-references within the questions ("it" in the 2nd question and 'this book' in the 3rd question), which is a good illustration of the capability of models to recognize stateful dialogs. This capability reflects well their top-ranking positions in the Hugging Face leaderboard.

However, even in this simplified test there are visible differences between the responses, which allow to evaluate the semantic quality and understandability of the provided texts.

For instance, 3 answers to the 1st question ('Is the Ulysses book written by James Joyce difficult to read?') differ a lot. Here, ChatGPT-4 and Claude-3 provide a focused, concise, readable text, while Gemini is in a verbose mood and produces much more text. It seems that Gemini tries to interpret this question as an intent to read this book,

and therefore adds at the end some advice concerning this potential next step: “If you're interested in tackling Ulysses, here are some tips: “ This is creative, interesting, and relevant to some extent, but in some situations it can also be annoying. For this reason, the score of Gemini is in Table 2 lowered to 4/5 for this question. This observation shows also the need for giving the model more hints or guidelines how it should respond, which is the subject of the next section.

Concerning Question 2 (“When was it written?”) the results show that the models interpret the intent of this question correctly, since such questions are usually not only about the writing period, but also about the publication date. Amazingly, all 3 models broaden the scope of their answers and provide the information about the writing period of the novel and also when and how it was published. However, the structure of Claude-3 reply is less optimal, compared to the other 2, and for this reason the score of this model is in this case 4/5.

The differences in Question 3 (Is it possible to summarize this book in no more than 150 words? Please try to do so.) are also very interesting. Although all 3 answers are quite optimal and informative, Gemini fails to mention that the novel has a structural link to Homer's “Odyssey”, while ChatGPT and Claude mention this. Therefore, in this test, the score of Gemini is a bit lower (4/5, Table 2).

From the evaluation results collected we conclude that ChatGPT -4, when used out-of-the-box, achieved in this test the best versatility and semantic performance. However, the other 2 models have also shown excellent semantic performance, and therefore the differences in Table 2 should rather be interpreted as negligible.

Second use case: Theory of Mind (ToM) scenarios

Obviously, there are a lot of domains and use cases where knowledge assistance capability of LLMs is undoubtedly useful such as science, history, philosophy, history, literature, physics, chemistry, law, publishing, and many more. Contrary to rather basic use for question answering and knowledge assistance presented in previous sections, more advanced capabilities emerge in LLMs due to their continuous evolution and growth.

One of such capabilities is scenario analysis based upon Theory of Mind (ToM). Recently, it was observed that previous, smaller models, published before November 2022, had no capability to solve ToM tests, and that this capability gradually emerged in more recent models. It is important to notice that, up-to-now, this capability was attributed only to humans. One of the most relevant works, which brought ToM into focus in the domain of Large Language Models is the paper (Kosinski, 2023a), which focused on quantifying these observations. In his work he states that:

> *'Eleven Large Language Models (LLMs) were assessed using a custom-made battery of false-belief tasks, considered a gold standard in testing Theory of Mind (ToM) in humans. The battery included 640 prompts spread across 40 diverse tasks, each one including a false-belief scenario, three closely matched true-belief control scenarios, and the reversed versions of all four. To solve a single task, a model needed to correctly answer 16 prompts across all eight scenarios. Smaller and older models solved no tasks, GPT-3-davinci-003 (from November 2022) and ChatGPT-3.5-turbo (from March 2023) solved 20, of the tasks, ChatGPT-4 (from June 2023) solved 75 of the tasks, matching the performance of six-year-old children observed in past studies. We explore the potential interpretation of these findings, including the intriguing possibility that ToM, previously considered exclusive to humans, may have spontaneously emerged as a byproduct of LLMs improving language skills.'*

The methodology of Theory of Mind has been developed in the past 30 years in the psychology of child development (Cherry, 2023) (Ruhl, 2023). As an example of a ToM test, we consider the following scenario, which was used among many other scenarios in the study of Kosinski (Kosinski, 2023b):

Scenario:

In the room, there are John, Mark, a cat, a box, and a basket. John takes the cat and puts it in the basket. He closes the basket. He leaves the room and goes to school. While John is away, Mark takes the cat

out of the basket and puts it in the box. He closes the box. Mark leaves the room and goes to work. John comes back home and wants to play with the cat.

Question:

Where will John look to take the cat?

Obviously, this is a very easy test to pass for anyone above the age of 5, but it can be challenging for children younger than 5, which may fail or pass this test depending on their age.

Third use case: image analysis

We focus now on implementing LLM-based image analysis, starting with an example image which shows an auditorium. The image was sent for analysis to the latest OpenAI LLM GPT-4o model, which was made accessible for public in May 2024. In the implementation, a modified Python script from Appending A was used, and the modification followed online documentation of OpenAI Vision API. The prompt and the reply of the model are presented below.

PROMPT

Please describe this image including color information.

REPLY

The image depicts an interior view of a lecture hall or auditorium. Here are the details:

- Seating: The seats are arranged in rows and are upholstered in a deep maroon color. Each seat has a light yellow writing surface attached to the back of the seat in front of it.

- Flooring: The floor is covered with a blue carpet that has small, light-colored square patterns.

- Walls: The walls are painted in a light beige or off-white color.

- Lighting: There are wall-mounted light fixtures that emit a soft, warm light, creating a well-lit environment.

- Door: There is an arched wooden door on the right side of the image. Above the door, there is an emergency exit sign with a green background and white text.

- Additional Seating: On the far right, there are a few seats with a greenish-blue upholstery, which appear to be different from the main seating area.

Overall, the image shows a clean and orderly lecture hall with a combination of maroon, yellow, blue, beige, and greenish-blue colors.

Evaluating the reply of GPT-4o LLM, we observe remarkably good accuracy of the description provided by the model concerning the colors and many details of the picture. The model lists all elements present in the room, and even succeeds with identifying the writing tables using the following description "Each seat has a light-yellow writing surface attached to the back of the seat in front of it". Moreover, it succeeds with deducing the function of the room as "a clean and orderly lecture hall with a combination of maroon, yellow, blue, beige, and greenish-blue colors" Therefore, this result is remarkably good.

References

Cherry K. (2023). "How the Theory of Mind Helps Us Understand Others". *Very Well Mind*. 4.04.2023. Https:// www.verywellmind.com/theory-of-mind-4176826

Grainger T. et al (2024). *AI-Powered Search*. Manning. ISBN 9781617296970

Kaczmarski P. & Vandamme F. (2024), "Implementing LLM-based AI -tools for knowledge assistance, text and scenario analysis, and image processing: case studies overview". *Communication & Cognition*, Vol.57 (3-4).

Kosinski, M. (2023a). *Theory of Mind May Have Spontaneously Emerged in Large Language Models*. Https://doi.org/10.48550/ arxiv.2302.02083

Kosinski, M. (2023b). *Project WIKI pages*. Https://osf.io/csdhb/wiki/home/

Lewis P. et al (2020). *Retrieval-Augmented Generation for Knowledge -Intensive NLP Tasks,* arXiv, 22 May 2020. Https://doi.org/10.48550/arXiv.2005.11401

OpenAI (2024). *Retrieval Augmented Generation (RAG) and Semantic Search for GPTs*. Https://help.openai.com/en/articles/8868588-retrieval-augmented-generation-rag-and-semantic-search-for-gpts

Ruhl C. (2023). "Theory of Mind in Psychology: People Thinking". 2023.08.08, *Simply Psychology*. Https://www.simplypsychology.org/theory-of-mind.html

Sarthi P. et al (2024). "RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval" (2024), *arXiv*, 31.01.2024. Https://doi.org/10.48550/arXiv.2401.18059

Wikipedia (2024a). *Retrieval-augmented generation*. Https://en.wikipedia.org/wiki/Retrieval-augmented_generation

Wikipedia (2024b). *Word2vec*. Https://en.wikipedia.org/wiki/Word2vec

About the authors

Peter Kaczmarski (PhD, MSc), BIKEMA (Babbage Institute for Knowledge and Information Management), Leiekaai 25-F, Gent, Belgium.

E-mail: peter.kaczmarski@bikema.org

Fernand J. Vandamme. Interuniversity Centre for Interdisciplinary Scientific Research (ICIWO vzw).

E-mail: f.vandamme@lotuswebtec.com